
Rakuten

Rakuten

- **Rakuten Group** has 1.2 billion members globally, and 130 million in the U.S., with more than 70 businesses across a variety of services, such as e-commerce, payment services, financial services, telecommunication, media, sports, adtech, etc.
- **Rakuten** is ranked the 5th largest e-commerce company in the world in 2018 (by revenue), and the 11th largest world wide internet service company (people).
- **Rakuten** has over 150+ Data Scientists of which 100 of them have a PhD in their respective field.
- With over 1 billion user events being stored daily in **Rakuten** servers across our variety of services, our data is at the heart of one of the most important strategies for the entire **Rakuten Group**.

Rakuten Sports Investments:
Golden State Warriors, FC Barcelona,
Spartan Race, and Tohoku Rakuten Golden Eagles, and Vissel Kobe

Rakuten Marketing

- **Rakuten Marketing** uses innovative, data-driven technology to help brands reach consumers with timely and authentic digital advertising experiences.
- **Rakuten Marketing** provides the largest affiliate marketing network world wide.
- **Rakuten Marketing** takes its pioneering AI and machine learning technology, pairs it with our unique data and inventory from the **Rakuten** ecosystem, and enables brands to identify new audiences and/or re-engage existing ones.
- This predictive technology allows **Rakuten Marketing** to deliver valuable experiences people love, resulting in cost-effective performance across its integrated marketing solutions: affiliate, display and search.
- **Rakuten Marketing** is headquartered in San Mateo, California, with offices in Australia, Singapore, Brazil, Japan, France, Germany, the United Kingdom and throughout the United States.

Spark Introduction, Exploratory Data Analysis W/ Pandas, and Building Spark ML Pipelines

Utah Data Engineering Meetup

November 14, 2018

Rakuten

This talk covers:

- **What Spark is?**
- **How does Spark work?**
- **Spark API's**
  - RDD
  - Dataframes
  - Datasets
- **Spark Transitions & Actions**
- **Spark ML**
- **Exploratory Data Analysis with PySpark, Pandas, and Jupyter Notebooks**
- **Building a ML Data Pipeline in PySpark**

References

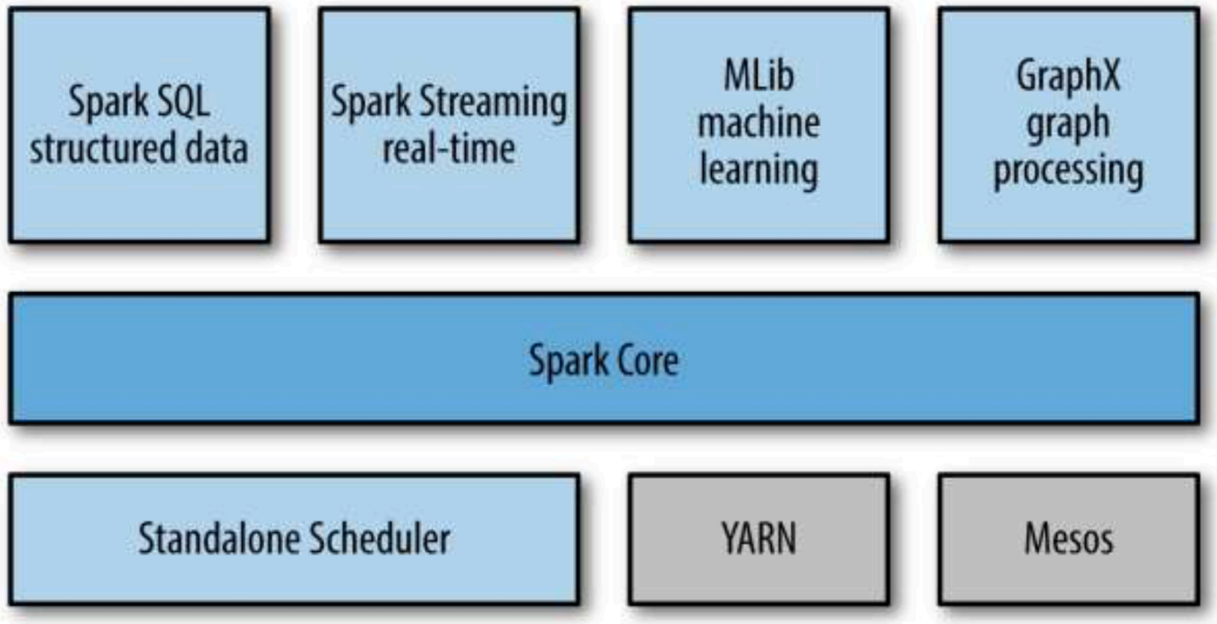
- Some slides on spark shamelessly taken from:
  - **Harvard University, Big Data Course** by Zoran B. Djordjevic, PhD
- These slides follow to a good measure the book:
  - **"*Learning Spark*"** by Holden Karau, Andy Konwinski, Patrick Wendell & Mathei Zaharia, O'Reilly 2015
- This talk also follows to a great measure the text:
  - ***"Machine Learning with Spark" Second Edition***, by Nick Pentreah, PAKT Publishing, 2017
- Another very good book on advanced Spark processing:
  - ***"Advanced Analytics with Spark"***, by Sandy Ryza et al., O'Reilly, 2015

What is Spark?

- **Spark is** designed to cover a wide range of computing workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming.
- **Spark extends** MapReduce model to efficiently support more types of computations, including interactive queries and stream processing.
- By supporting these workloads in the same engine, **Spark makes** it easy and inexpensive to *combine* different processing types.
- **Spark is** designed to be highly accessible, offering simple APIs in Python, Java, Scala, R and SQL, and rich built-in libraries.
- **Spark integrates** closely with other Big Data tools.
- **Spark can** run in Hadoop clusters and access any Hadoop data source, including NoSQL DBs like Cassandra.

What is Spark good for?

- **Spark is** a fast *general* processing engine for large scale data processing
- **Spark is** designed for iterative computations and interactive data mining.
- **Spark supports** use of well known languages: Scala, Python, Java and R
- With Spark streaming the same code could be used on data at rest and on data in motion
- **Spark has** several key modules:



Key Modules

- **Spark Core contains** the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and others.
- **Spark Core is** the home to the API that defines *resilient distributed datasets* (RDDs), which are Spark's main programming abstraction.
- **RDDs represent** a collection of items distributed across many compute nodes that can be manipulated in parallel. In recent releases less emphasis is placed on RDDs and more and DataFrames and Datasets.
- **Spark SQL is** Spark's package for working with structured data.
- **Spark SQL allows** querying data via SQL as well as the Apache Hive variant of SQL—Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON.
- **Spark SQL allows** developers to intermix SQL queries with the programmatic data manipulations supported by RDDs and DataFrames in Python, Java, and Scala, all within a single application, thus combining SQL with complex analytics.

Key Modules

- **Spark Streaming is** a Spark component that enables processing of live streams of data. Data streams include log files of web servers, or queues of messages.
- **Spark Streaming API** for manipulating data streams closely matches the Spark Core's RDD API, making it easy to move between apps that manipulate data in memory, on disk, or arriving in real time.
- **Spark Streaming** provides the same degree of fault tolerance, throughput, and scalability as Spark Core.
- **MLlib is** a library containing common machine learning (ML) functionality.
- **MLlib provides** multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import.
- **MLlib provides** some lower-level ML primitives, including a generic gradient descent optimization algorithm.
- All of ML methods are designed to scale out across a cluster.

Three API-s, RDD, DataFrames & Datasets

- In Spark 2.x, there are three sets of APIs—**RDDs, DataFrames, and Datasets.**
- We need to understand why and when to use each set;
  - outline their performance and optimization benefits;
  - enumerate scenarios when to use **DataFrames and Datasets instead of RDDs**.
- In Apache Spark 2.0 DataFrames and Datasets APIs are unified

How does Spark work?

- **Spark is** an *open-source* software solution that performs rapid calculations on *in-memory distributed datasets.*
- In-memory distributed datasets are referred to as RDDs
- ***RDDs are*** *Resilient Distributed Datasets*
  - **RDD is** the key Spark concept and the basis for what Spark does
  - **RDD is** a distributed collections of objects that can be cached in memory across cluster and can be manipulated in parallel.
  - **RDD could** be automatically recomputed on failure
  - **RDD is** resilient – can be recreated on the fly from known state
  - Immutable – already defined **RDDs can** be used as a basis to generate derivative RDDs but are never mutated
  - Distributed – the dataset is often partitioned across multiple nodes for increased scalability and parallelism

How does Spark Work?

- **RDD**
  - Your data is loaded in parallel into structured collections
- **Actions**
  - Manipulate the state of the working model by forming new RDDs and performing calculations upon them
- **Persistence**
  - Long-term storage of an RDD’s state
- **Spark Application is a definition in code of**
  - RDD creation
  - Actions
  - Persistence
- **Spark Application results** in the creation of a DAG (Directed Acyclic Graph)
- Each DAG is compiled into stages
- Each Stage is executed as a series of Tasks
- Each Task operates in parallel on assigned partitions

Spark DataFrames

- Like an RDD, a **DataFrame is** an immutable distributed collection of data.
- Unlike an RDD, **data is** organized into named columns, like a table in a relational database.
- Designed to make large data sets processing even easier, **DataFrame allows** developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction; it provides a domain specific language API to manipulate your distributed data; and makes Spark accessible to a wider audience, beyond specialized data engineers.
- In **Spark 2.0, DataFrame APIs merged** with Datasets APIs, unifying data processing capabilities across libraries.
- Because of this unification, developers now have fewer concepts to learn or remember, and work with a single high-level and type-safe API called Dataset.

RDDs, When to Use Them

- **RDD was** the primary user-facing API in Spark since its inception. At the core, an RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions.
- **Common use cases for using RDDs are:**
  - you want low-level transformation and actions and control on your data;
  - your data is unstructured, such as media streams or streams of text;
  - you want to manipulate your data with functional programming constructs rather than domain specific expressions;
  - you don't care about imposing a schema, such as columnar format, while processing or accessing data attributes by name or column; and
  - you can forgo some optimization and performance benefits available with DataFrames and Datasets for structured and semi-structured data.
- Are RDDs being relegated as second class citizens? Are they being deprecated?
  - The answer is a **NO!**
- We can seamlessly move between DataFrame or Dataset and RDDs at will—by simple API method calls—and DataFrames and Datasets are built on top of RDDs.

DataFrames

- A **DataFrame** is a table of data with rows and columns. The list of columns and the types in those columns is the schema.
- A simple analogy to Spark **DataFrame** is a spreadsheet with named columns. The fundamental difference is that while a spreadsheet sits on one computer in one specific location, a Spark **DataFrame** can span thousands of computers.
- One is putting the data on more than one computer because either the data is too large to fit on one machine or it would simply take too long to perform that computation on one machine.
- The **DataFrame** concept is not unique to Spark. The concept is borrowed from R. Python has similar concepts. However, Python/R **DataFrames** (with some exceptions) exist on one machine rather than multiple machines. This limits what you can do with a given DataFrame in Python and R to the resources that exist on that specific machine.
- Spark has language interfaces for both Python and R, and it is easy to convert to Spark **DataFrames** to Pandas (Python) **DataFrames**

Ease-of-use of APIs with structure

- When data are rendered as a ***DataFrame***, it is much simpler to perform ***agg***, ***select***, ***sum***, ***avg***, ***map***, ***filter***, or ***groupBy*** operations by accessing a Dataset typed object's Device IoT Data than using RDD rows' data fields.
- You could extract all those data from an RDD using some kind of regular expression process but that is much more tedious.

Transformations and Actions

- ***Transformations*** construct a new RDD/Dataframe from a previous one. For example, one common transformation is filtering data that matches a predicate. (.filter())
- ***Actions*****,** on the other hand, compute a result based on an RDD/Dataframe, and either return it to the driver program or save it to an external storage system (e.g., OS or HDFS).

Examples, Actions

- *Basic actions on an RDD containing numbers {1, 2, 3, 3}*

Function name	Purpose	Example	Result
collect()	Return all elements from the RDD.	rdd.collect()	{1, 2,3,3}
count()	Return all elements from the RDD.	rdd.count()	4
countByValue()	Number of times each element occurs in the RDD.	rdd.countByValue()	{(1,1),(2,1),(3,2)}
take(num)	Return num elements from the RDD.	rdd.take(2)	{1,2}
top(num)	Return the top num elements the RDD.	rdd.top(2)	{3,3}
takeOrdered(num)(ordering)	Return num elements based on provided ordering.	rdd.takeOrdered(2)(myOrdering)	{3,3}
takeSample(withReplacement,num,[seed])	Return num elements at random.	rdd.takeSample(false, 1)	nondeterministic
reduce()	Combine the elements of the RDD together in parallel (e.g., sum).	rdd.reduce((x, y) => x + y)	9
fold(zero)(func)	Same as reduce() but with the provided zero value.	rdd.fold(0)((x, y) => x + y)	9
aggregate(zeroValue)(seqOp, combOp)	Similar to reduce() but used to return a different type.	rdd.aggregate((0, 0)) ((x, y) => (x._1 + y, x._2 + 1), (x, y) => (x._1 + y._1, x._2 + y._2))	(9,4)
foreach(func)	Apply the provided function to each element of the RDD.	rdd.foreach(func)	

Examples, Transformations

- Since pair RDDs contain tuples, we need to pass functions that operate on tuples rather than on individual elements.
- *Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})*

Function name	Purpose	Example	Result
reduceByKey(func)	Compbine values with the same key	rdd.reduceByKey(x,y) => x + y	{(1,2),(3,10)}
groupByKey()	Group values with the same key	rdd.groupByKey()	{(1,[2]),(3,[4,6])}
mapValues(func)	Apply a function to each value of a pair RDD without changing the key.	rdd.mapValues(x => x+1)	{(1, 3), (3, 5), (3,7)}
flatMapValues(func)	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	rdd.flatMapValues(x => (x to 5)	{(1,2), (1,3), (1,4), (1, 5), (3, 4), (3,5)}
keys()	Return an RDD of just the keys.	rdd.keys()	{1,3,5}
values()	Return an RDD just of values	rdd.values()	{2,4,6}
sortByKey()	Return and RD sorted by the key	Rdd.sortByKey()	{(1,2),(3,4)(3,6)}

Transformations on two pair RDD

Two pair *RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)})*

Function name	Purpose	Example	Result
substractByKey	Remove elements with a key present in the other RDD	Rdd.substractByKey(other)	{(1,2)}
join	Perform an inner join between two RDDs	Rdd.join(other)	{(3,(4,9)),(3,(6,9))}
rightOuterJoin	Perform a join between two RDDs where the key must be present in the first RDD	Rdd.rightOuterJoin(other)	{(3,(Some(4),9)), (3,(Some(6),9))}
leftOuterJoin	Perform a join between two RDDs where the key must be present in the other RDD.	rdd.leftOuterJoin(other)	{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}
cogroup	Group data from both RDDs sharing the same key.	rdd.cogroup(other)	{(1,([2],[])), (3, ([4, 6],[9]))}

- Sometimes working with pairs can be awkward if we want to access only the value part of our pair RDD.
- Since this is a common pattern, Spark provides the mapValues(func) function, which is the same as map{case (x, y): (x, func(y))}.

Aggregations

- When datasets are described in terms of key/value pairs, it is common to want to aggregate statistics across all elements with the same key. We have looked at the fold(), combine(), and reduce() actions on basic RDDs, and similar per-key transformations exist on pair RDDs.
- Spark has a similar set of operations that combines values that have the same key. These operations return RDDs and thus are transformations rather than actions.
- reduceByKey() is quite similar to reduce(); both take a function and use it to combine values. reduceByKey() runs several parallel reduce operations, one for each key in the dataset, where each operation combines values that have the same key. Because datasets can have very large numbers of keys, reduceByKey() is not implemented as an action that returns a value to the user program. Instead, it returns a new RDD consisting of each key and the reduced value for that key.
- foldByKey() is quite similar to fold(); both use a zero value of the same type of the data in out RDD and combination function.

Spark SQL

- Spark SQL used to be a separate API in Spark 1.x. Spark SQL engine is still there and implements most features of SQL 2003 standard and many sophisticated add-on.
- Spark SQL Engine is for the most part hidden behind DataFrame and Dataset API.
- Spark SQL uses this extra information to perform extra optimizations. There are several ways to interact with Spark SQL including SQL, the DataFrames API and the Datasets API.
- One use of Spark SQL is to execute SQL queries written using either a basic SQL syntax or HiveQL (Hive is a data warehouse application built atop of Hadoop)
- Spark SQL can also be used to read data from an existing Hive installation. When running SQL from within another programming language the results will be returned as a DataFrame.
- You can also interact with the SQL interface using the command-line or JDBC/ODBC.
- SparkSQL could read and write data from Parquet files, JSON files, Hive, Cassandra

Machine Learning Pipeline with Spark

Utah Data Engineering Meetup

November 14, 2018

Rakuten

Need for Machine Learning System with Spark

1. The scale of data that needs to be analyzed at many large companies means that humans only analysis of data quickly becomes infeasible as the business grows.
2. If they could write software that would do everything the humans do, they would get rid of humans. Humans are so expensive and difficult.
3. Model-driven approaches such as machine learning and statistics can often benefit from uncovering patterns that cannot be seen by humans (due to the size and complexity of the data sets)
4. Model-driven approaches can avoid human and emotional biases (as long as the correct processes are carefully applied)

Spark MLLib

- **MLlib is** Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy.
- **MLlib consists** of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.
- Spark's Machine Learning API until Spark 2.0 divided into two packages:
  - spark.mllib which contained the original API built on top of RDDs.
  - spark.ml which provided higher-level API built on top of DataFrames for constructing ML pipelines.
- Using **spark.ml is recommended** because with DataFrames the API is more versatile and flexible.
- Spark 2.0 apparently deemphasized division between the two API-s and treats spark.ml library as a part of spark.mllib
- **Spark is** open source and you could/should contribute new algorithms to spark.mllib.

Some MLlib Supported Features and Algorithms

- Basic statistics
  - Summary statistics and correlations
  - Stratified sampling and hypothesis testing
  - Streaming significance testing
  - Random data generation
- Classification and regression
  - Linear models (SVMs, logistic regression, linear regression)
  - Naive bayes
  - Decision trees
  - Ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - Isotonic regression
- Collaborative filtering
  - Alternating least squares (ALS)
- Clustering
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Latent Dirichlet allocation (LDA)
  - Bisecting k-means and streaming k-means
- Dimensionality reduction
  - Singular value decomposition (SVD)
  - Principal component analysis (PCA)
- Feature extraction and transformation

For small data sets use other packages

- If your datasets are small or moderate in size, you are better of with one of many Machine Learning toolkits:
- Scikit-learn is very comprehensive Python Machine Learning toolkit. Tool for hackers.
- Matlab can do anything that smacks of Math and Statistics. Large number of implemented ML algorithms, excellently integrated graphing utilities.
- R – your favorite. Does not scale as well as MLLib but has a much larger number of implemented algorithms
- WEKA is Java
- Rapid Miner is Java
- Apache Mahout is Java and uses Spark in the background. Used to be very popular.
- Microsoft Azure ML is an excellent platform that scales well.
- Accord.MachineLearning is a .Net package
- Vopal Wabbit is C++
- MultiBoost and Shogun are C++

When to use Spark ML

- If you are dealing with small samples classical API-s are faster and typically easer to deal with.
- Matlab or R are fully integrated testing and modelling suites. If you use Matlab or R on small data sets you will finish your job much more quickly and can do it on any machine, Windows, Mac, Linux. There is a free version of Matlab called Octave.
- If you are processing large volumes of data in Spark Batch or Spark Streaming mode and need to perform some fitting or classification (any ML algorithm) as an addition to other processing, it is convenient to add that particular part of analysis in Spark ML API. You deal with the same API, you do not need separate infrastructure and so on..
- If you have standalone ML type task, e.g. you need to classify, cluster or make estimates over very large volumes of data, there are other technologies, Neural Networks in particular, that might perform that task better, with higher accuracy and at a higher speed than Spark ML.

Exploratory Data Analysis with Pandas and Jupyter Notebooks

Rakuten

Conclusion… Kind of…

- Spark provides a sophisticated if not necessarily intuitive API for standard Machine Learning tasks.
- Spark's for Machine Learning MLlib is evolving and gaining both:
  - Performance
  - Convenience (hopefully)
- SparkMLlib is not the only game in town, but at the moment appears very promising.

Questions?

Slides & Code:
https://github.com/stirlingw/utah-data-engineering-pyspark

Rakuten